Stat 243

Base Graphics (part 2)

Gaston Sanchez

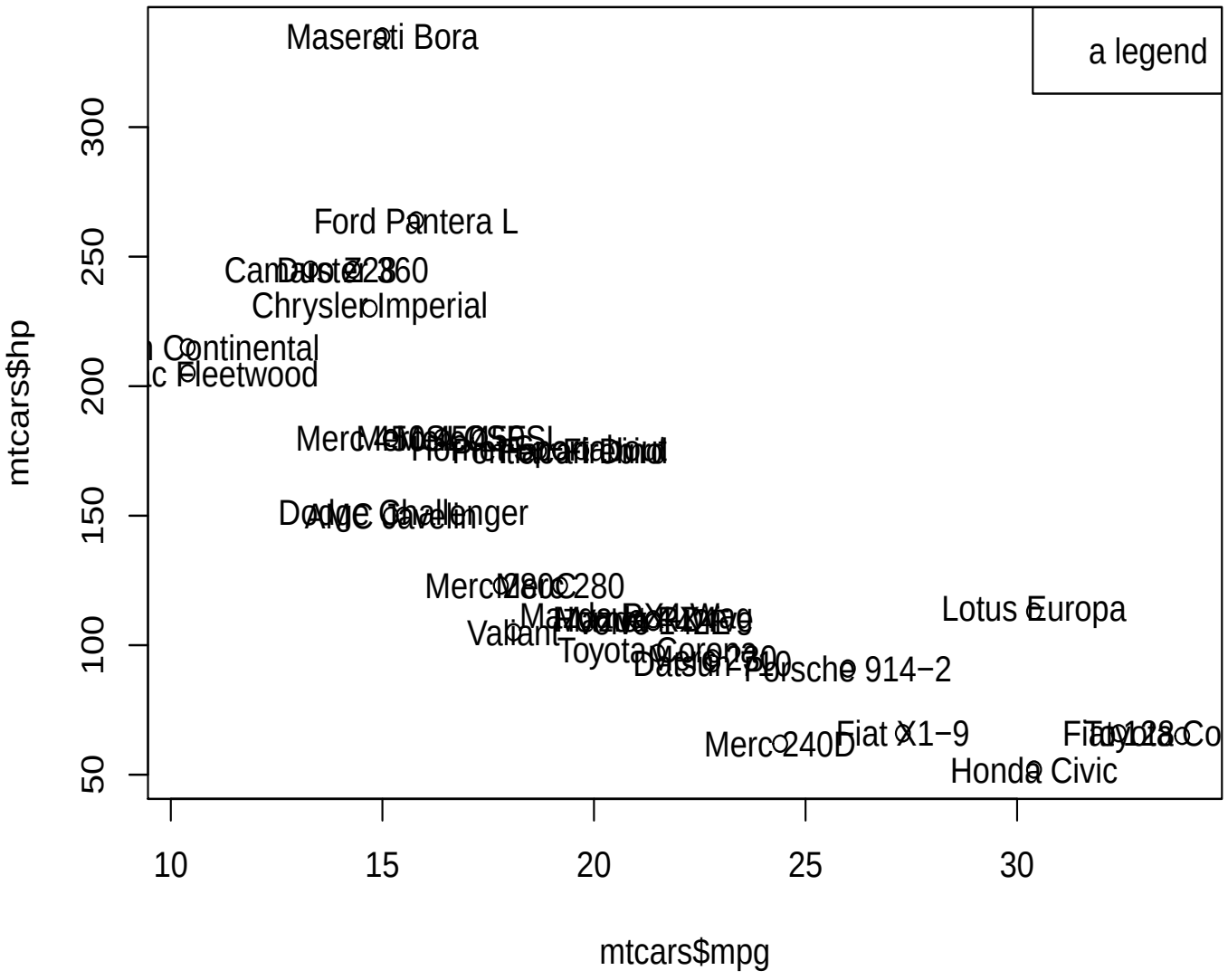
Low-Level Functions

High and Low level functions

- Usually we call a high-level function
- Most times we change the default arguments
- Then we call low-level functions

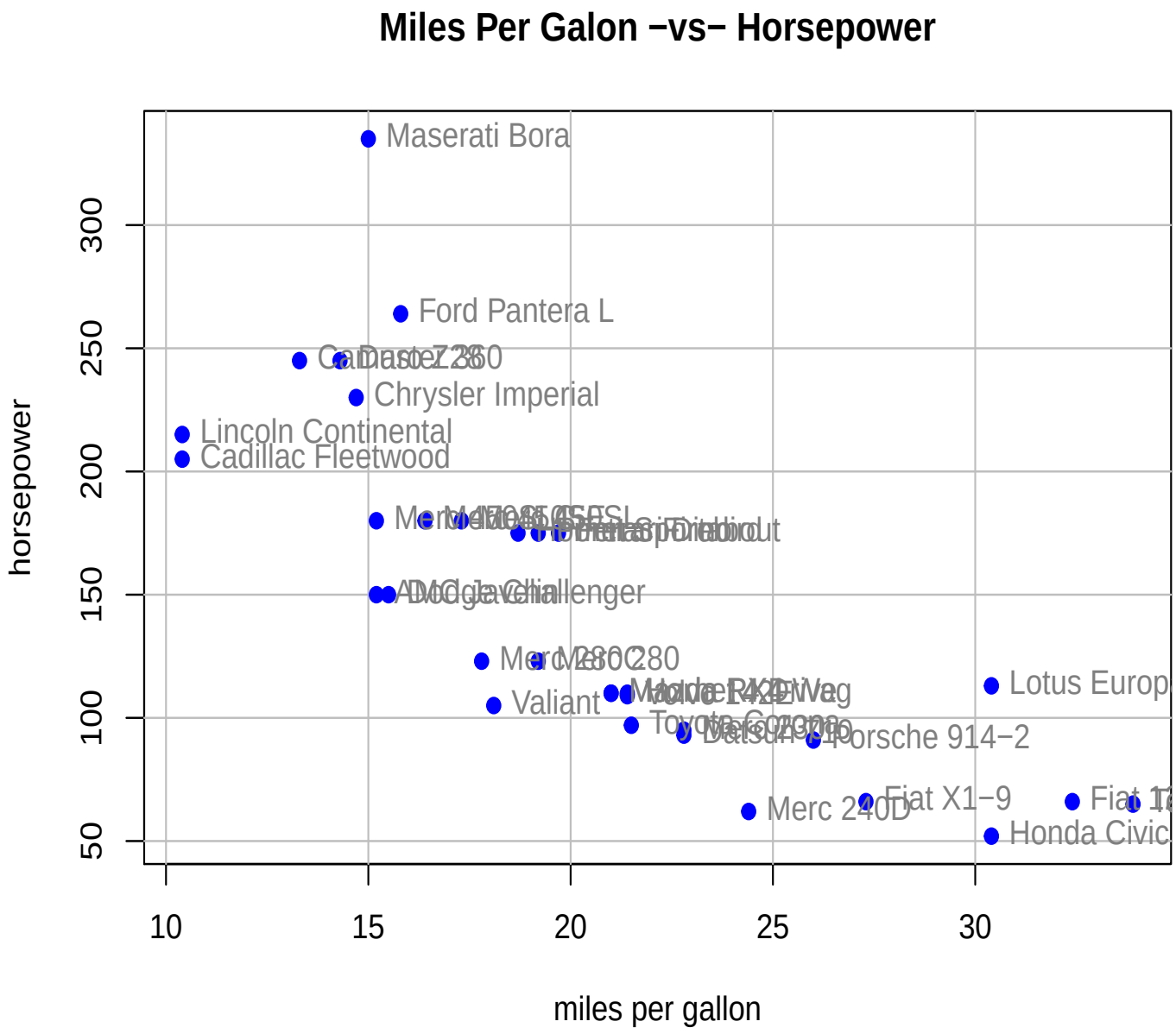
Scatter plot

```r
# simple scatter-plot
plot(mtcars$mpg, mtcars$hp)

# adding text
text(mtcars$mpg, mtcars$hp, labels = rownames(mtcars))

# dummy legend
legend("topright", legend = "a legend")

# graphic title
title("Miles Per Galon -vs- Horsepower")
```


Miles Per Galon −vs− Horsepower
a legend
Maserati Bora
Ford Pantera L
Chrysler Imperial
Continental
Fleetwood
Merc 280
Lotus Europa
Porsche 914−2
Fiat X1−9
Merc 240D
Honda Civic
mtcars$hp
mtcars$mpg
50
100
150
200
250
300
10
15
20
25
30

Scatter plot

```
# simple scatter-plot
plot(mtcars$mpg, mtcars$hp, type = "n",
     xlab = "miles per gallon", ylab = "horsepower")
# grid lines
abline(v = seq(from=10, to=30, by=5), col = 'gray')
abline(h = seq(from=50, to=300, by=50), col = ' gray')
# plot points
points(mtcars$mpg, mtcars$hp, pch = 19, col = "blue")
# plot text
text(mtcars$mpg, mtcars$hp, labels = rownames(mtcars),
     pos = 4, col = "gray50")
# graphic title
title("Miles Per Galon -vs- Horsepower")
```


Miles Per Galon −vs− Horsepower
horsepower
miles per gallon
50
100
150
200
250
300
10
15
20
25
30
Maserati Bora
Ford Pantera L
Chrysler Imperial
Lincoln Continental
Cadillac Fleetwood
Valiant
Lotus Europ
Porsche 914−2
Fiat X1−9
Merc 240D
Honda Civic

Low-level functions

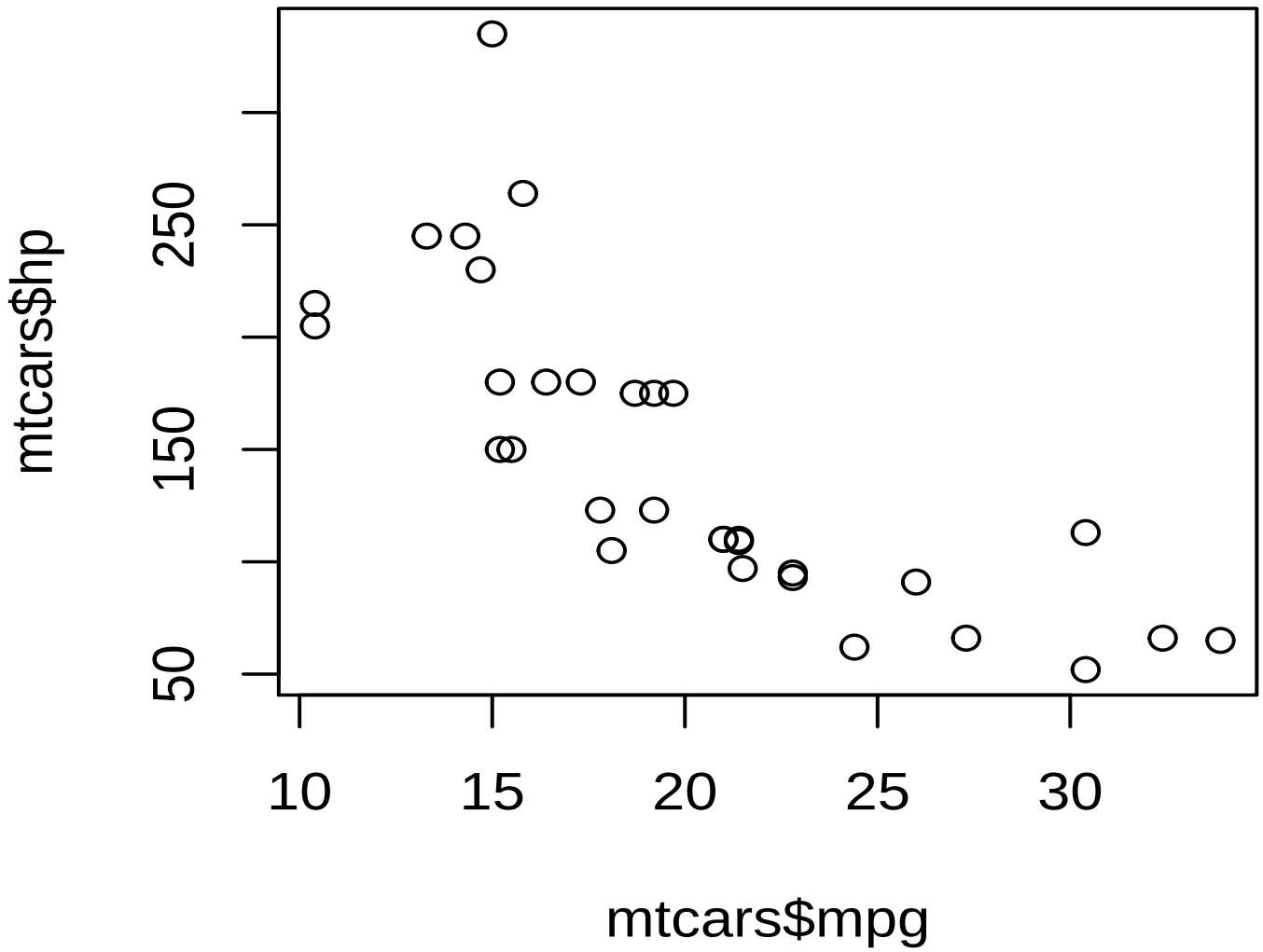
Function	Description
`points()`	points
`lines()`	connected line segments
`abline()`	straight lines across a plot
`segments()`	disconnected line segments
`arrows()`	arrows
`rect()`	rectangles
`polygon()`	polygons
`text()`	text
`symbols()`	various symbols
`legend()`	legends

Drawing Points with `points()`

```
points(x, y, pch = int, col = str)
```

- `pch` integer or string indicating type of point character
- `col` color of points

```r
# drawing points
plot(mtcars$mpg, mtcars$hp, type = "n")
points(mtcars$mpg, mtcars$hp)
```

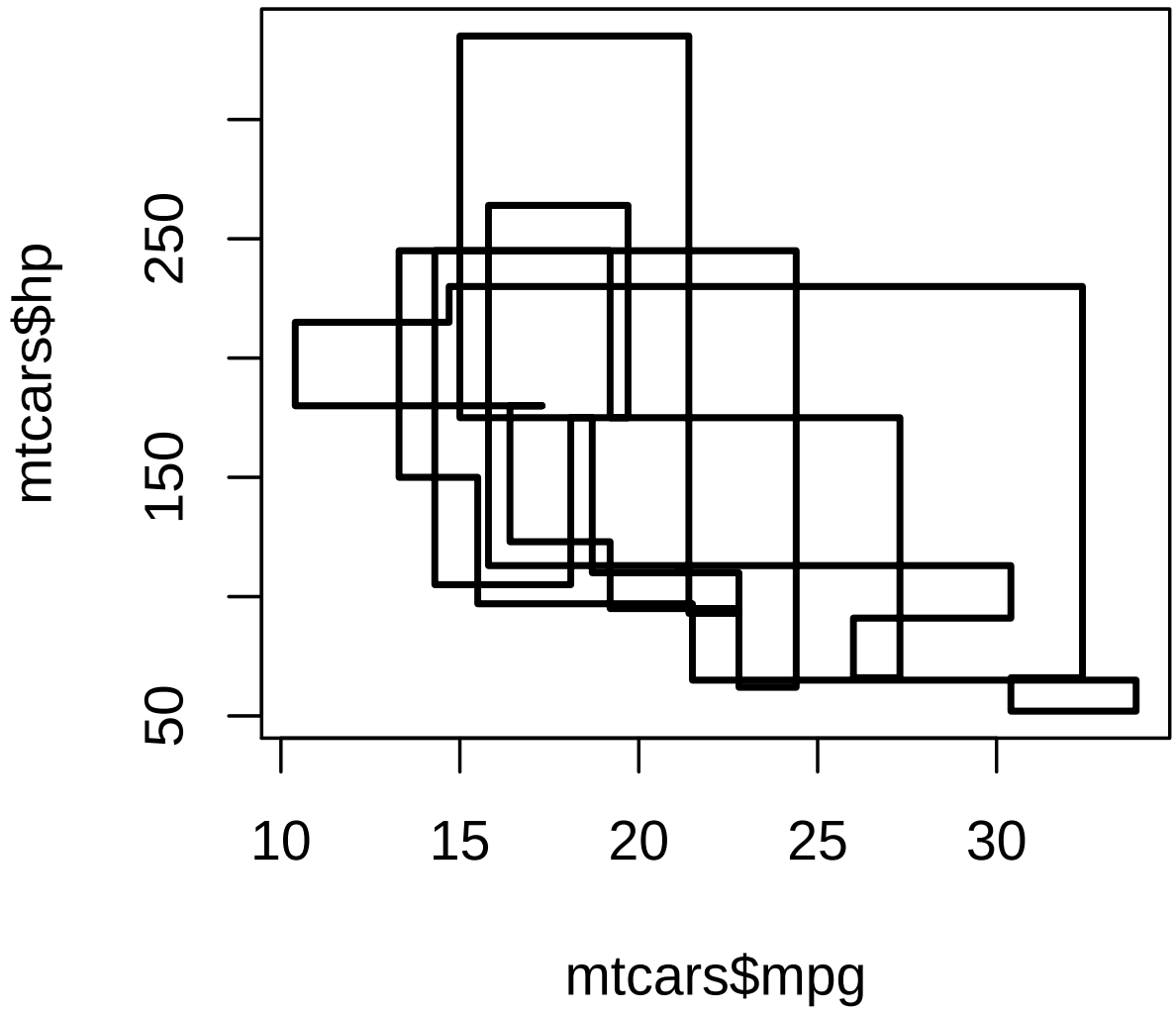


Connected Line Segments

```
lines(x, y, lty = str, lwd = num, col = str)
```

- `lty` specifies the line texture.
- It should be one of `"blank"` (0), `"solid"` (1), `"dashed"` (2), `"dotted"` (3), `"dotdash"` (4), `"longdash"` (5) or `"twodash"` (6).
- `lwd` and `col` specify the line width and color

```r
# connected lines
plot(mtcars$mpg, mtcars$hp, type = "n")
lines(mtcars$mpg, mtcars$hp, type = "s", lwd = 2)
```

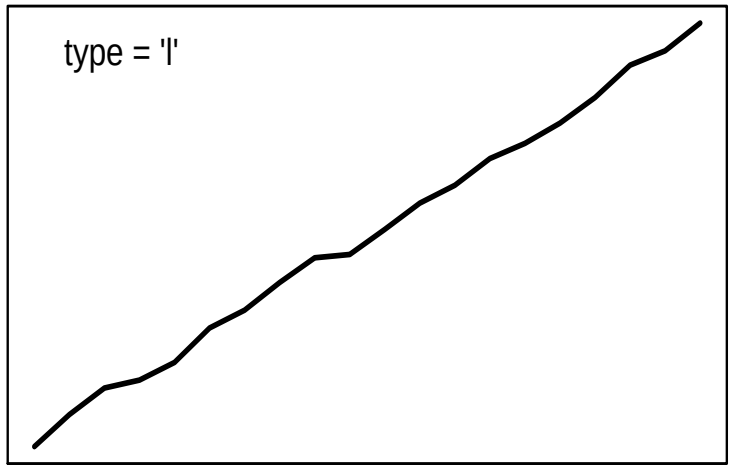


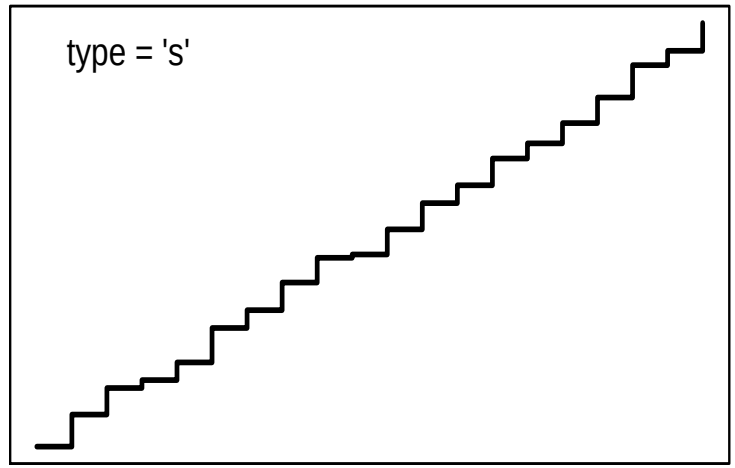
Line Graph Options

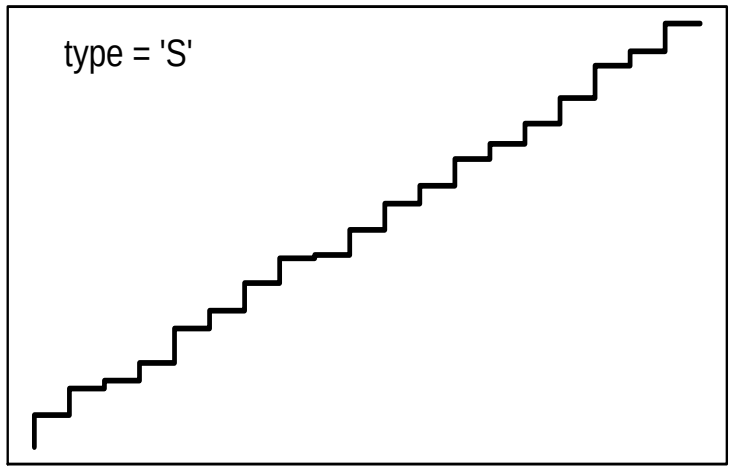
The `type` argument can be used to produced other types of lines

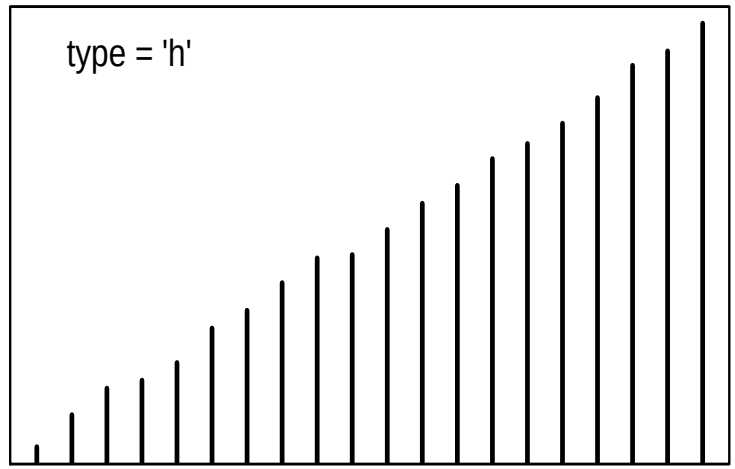
- `type = "l"` line graph
- `type = "s"` step function (horizontal first)
- `type = "S"` step function (vertical first)
- `type = "h"` high density (needle) plot
- `type = "p"` draw points
- `type = "b"` draw points and lines
- `type = "o"` over-plotting points and lines

Line Graph Options





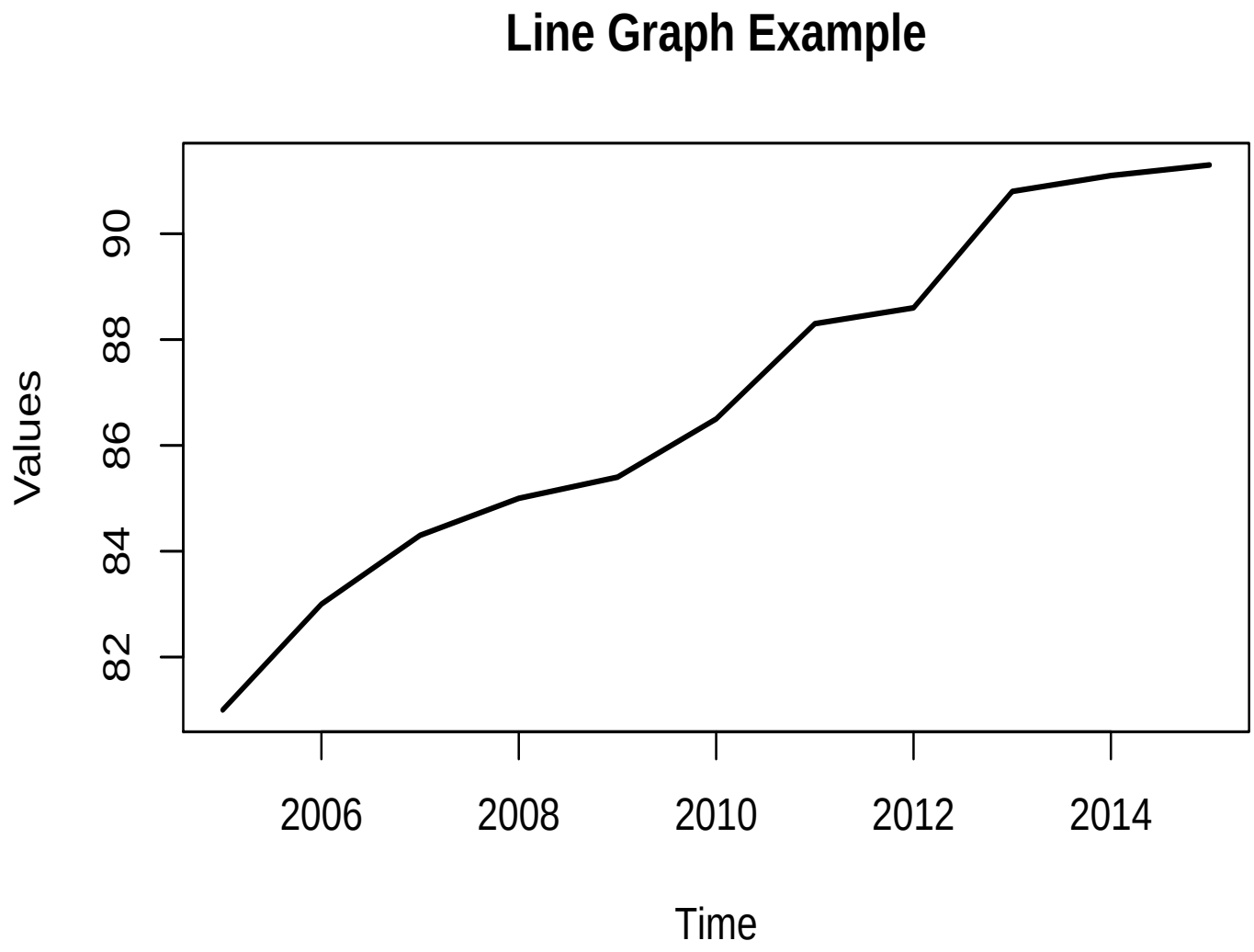




Connected Line Segments

```
x <- 2005:2015
y <- c(81, 83, 84.3, 85, 85.4, 86.5, 88.3,
       88.6, 90.8, 91.1, 91.3)

plot(x, y, type = 'n', xlab = "Time", ylab = "Values")
lines(x, y, lwd = 2)
title(main = "Line Graph Example")
```

Connected Line Segments

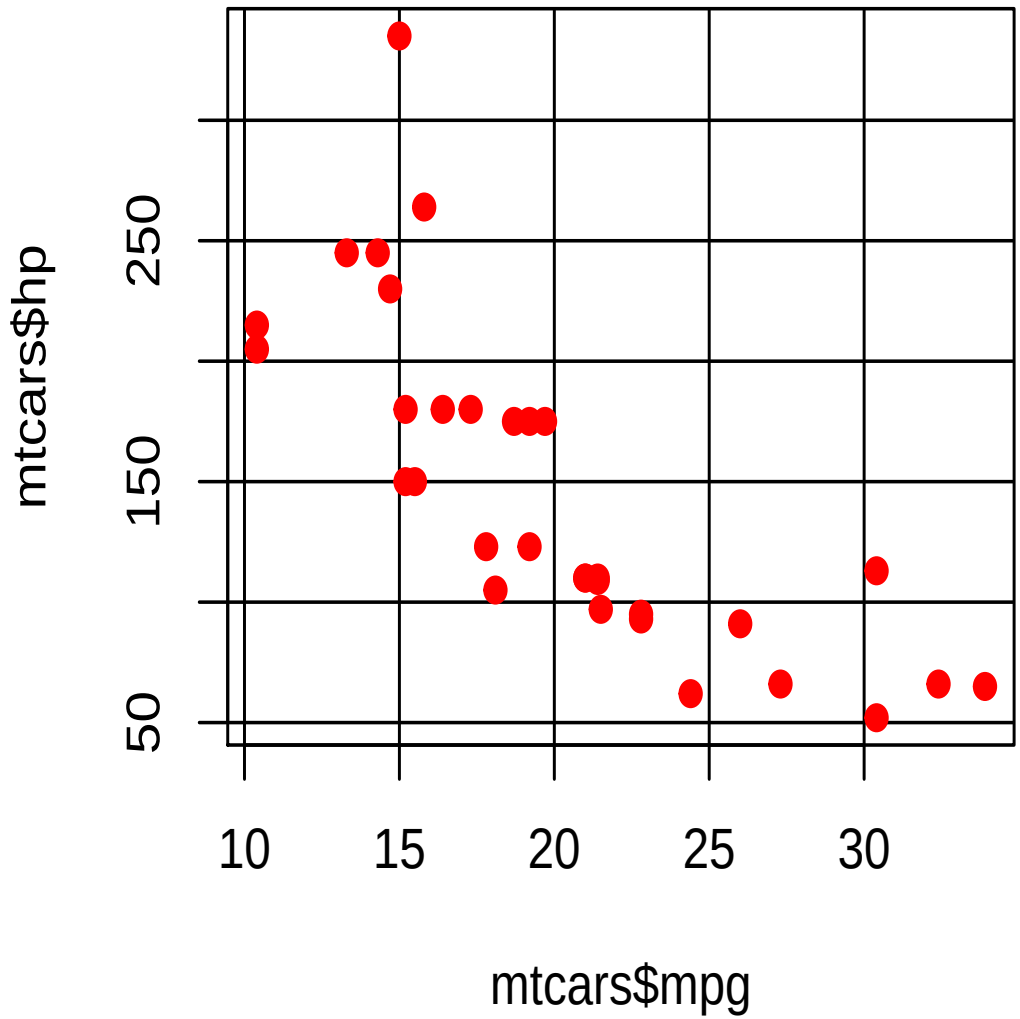


Drawing Straight Lines

```
abline(a = intercept, b = slope)
abline(h = numbers)
abline(v = numbers)
```

- The `a / b` form specifies a line in intercept / slope form
- `h` specifies horizontal lines at given y-values
- `v` specifies vartical lines at given x-values

```r
# drawing straight lines
plot(mtcars$mpg, mtcars$hp, type = "n")
abline(v = seq(10, 30, by = 5), h = seq(50, 300, by = 50))
points(mtcars$mpg, mtcars$hp, pch = 19, col = "red")
```



Drawing Disconnected Lines

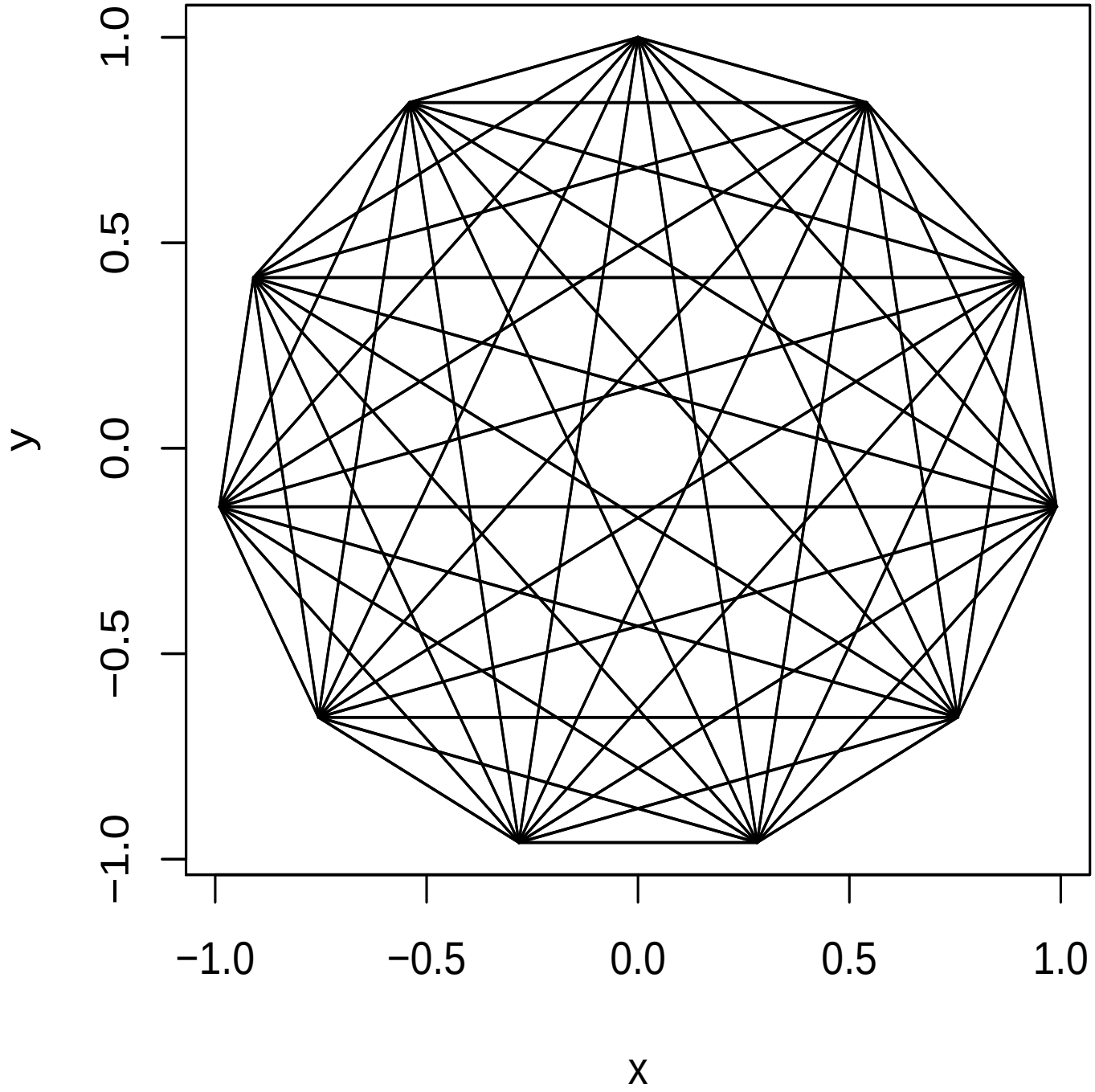
Disconnected lines can be drawn with the function:

```
segments(x0, y0, x1, y1)
```

- The `x0, y0, x1, y1` arguments give the start and end coordinates of the segments.
- Line texture, colour and width arguments can also be given.

Drawing Line Segments

```
n <- 11
theta <- seq(0, 2 * pi, length = n + 1)[1:n]
x <- sin(theta)
y <- cos(theta)
v1 <- rep(1:n, n)
v2 <- rep(1:n, rep(n, n))

plot(x, y, type = 'n')
segments(x[v1], y[v1], x[v2], y[v2])
```


1.0
0.5
0.0
−0.5
−1.0
y
−1.0
−0.5
0.0
0.5
1.0
x

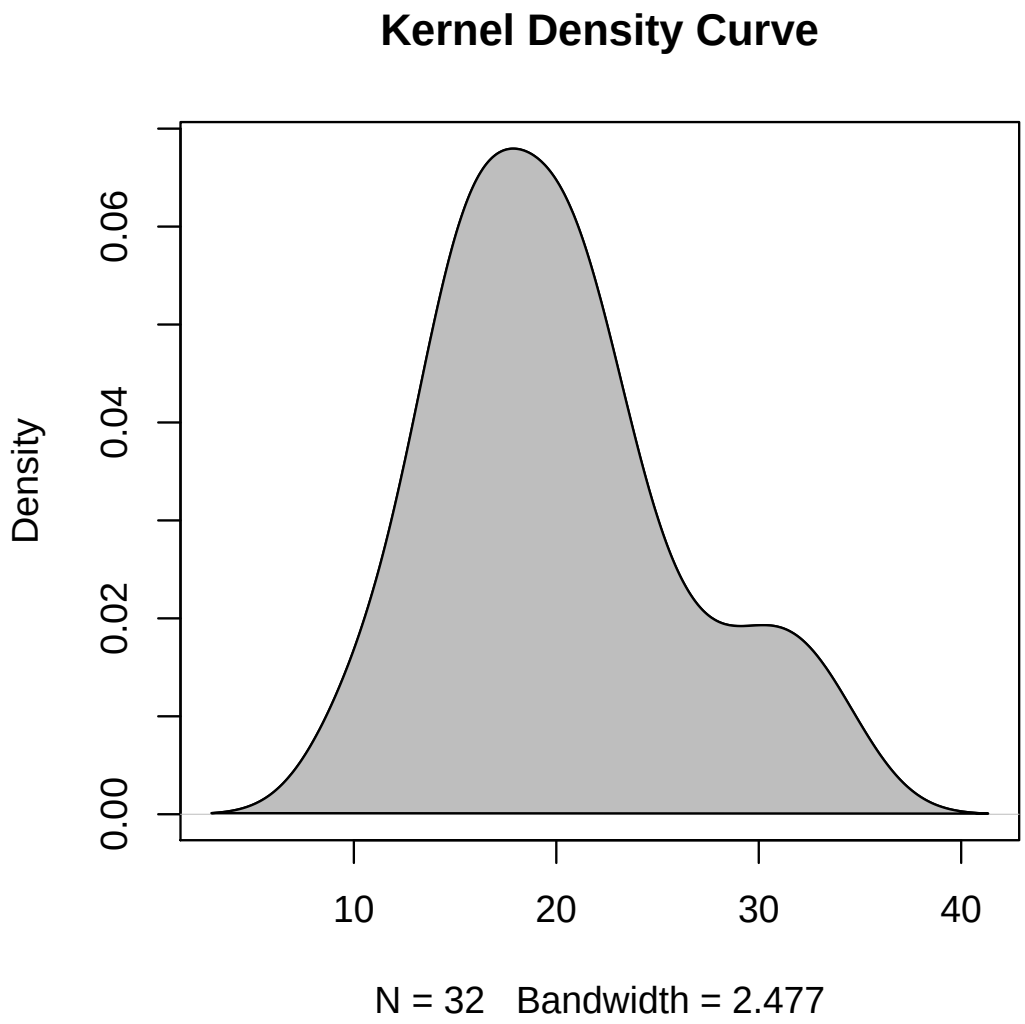
Drawing Polygons}

Polygons can be drawn with the function:

```
polygon(x, y, col = str, border = str)
```

- `x, y` give the coordinates of the polygon vertexes. `NA` values separate polygons.
- `col` specifies the color of the interior.
- `border` specifies the color of the border.
- line texture and width specifications can also be given.

Drawing Polygons



Adding Text

We can add text using the function:

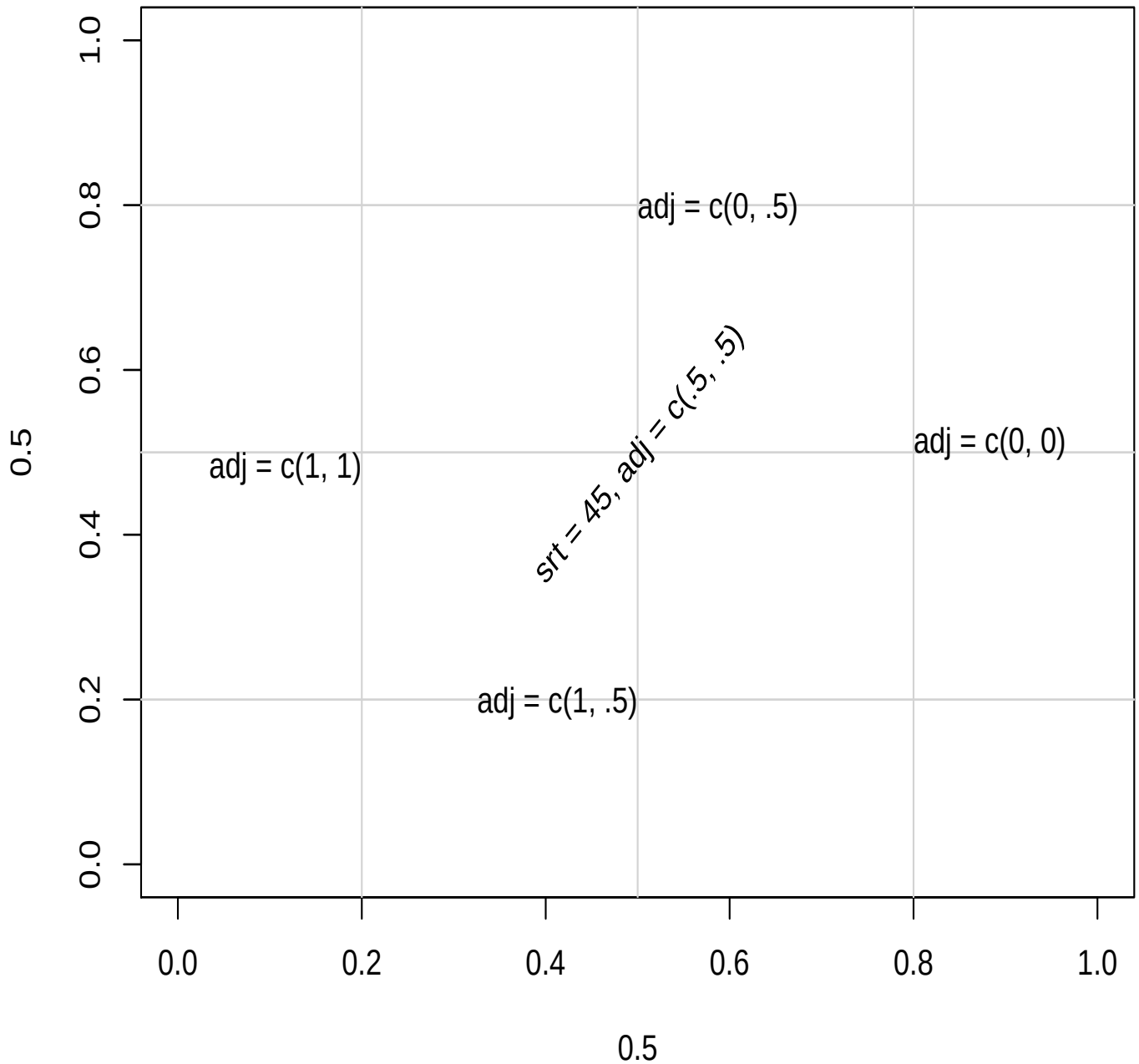
```
text(x, y, labels, ...)
```

- `x, y` give the coordinates of the text.
- `labels` gives the actual text strings.
- `font` optional font for the text.
- `col` optional color for the text.
- `srt` rotation of the text.
- `adj` justification of the text.

Drawing Text

```
plot(0.5, 0.5, xlim = c(0, 1), ylim = c(0, 1), type = 'n')
abline(h = c(.2, .5, .8),
       v = c(.5, .2, .8), col = "lightgrey")
text(0.5, 0.5, "srt = 45, adj = c(.5, .5)",
     srt = 45, adj = c(.5, .5))
text(0.5, 0.8, "adj = c(0, .5)", adj = c(0, .5))
text(0.5, 0.2, "adj = c(1, .5)", adj = c(1, .5))
text(0.2, 0.5, "adj = c(1, 1)", adj = c(1, 1))
text(0.8, 0.5, "adj = c(0, 0)", adj = c(0, 0))
```

Drawing Text



Adding a Legend

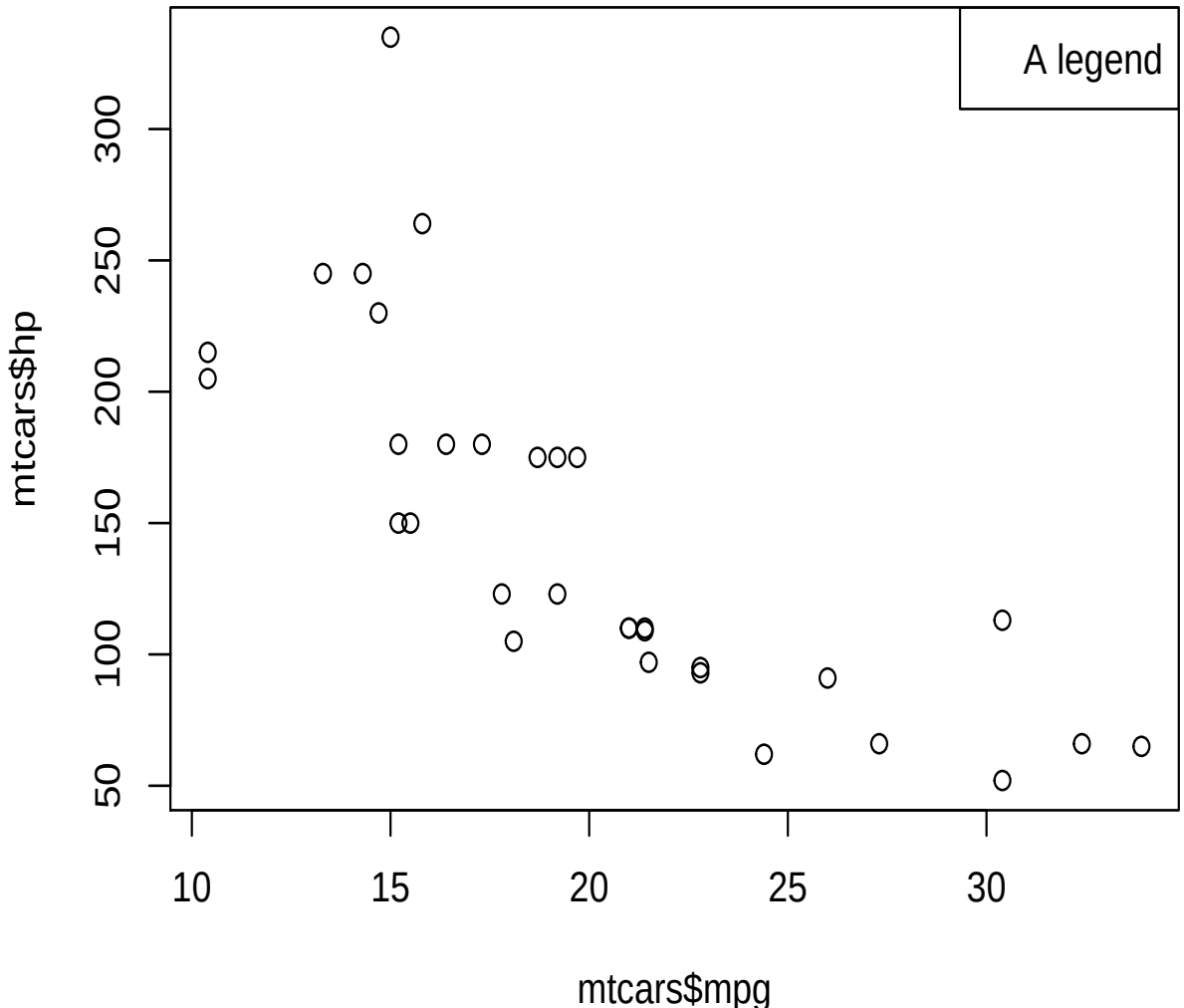
Legends can be added with:

```
legend(xloc, yloc, legend = text
       lty = linetypes, lwd = linewidths,
       pch = glyphname, col = colours,
       xjust = justification, yjust = justification)
```

- `xloc` and `yloc` give the coordinates where the legend is to be placed
- `xjust` and `yjust` give the justification of the legend box with respect to the location.

Adding Legends

```
# coords of exact line
plot(mtcars$mpg, mtcars$hp)
legend("topright", legend = "A legend")
```



Plots from scratch

General Recipe

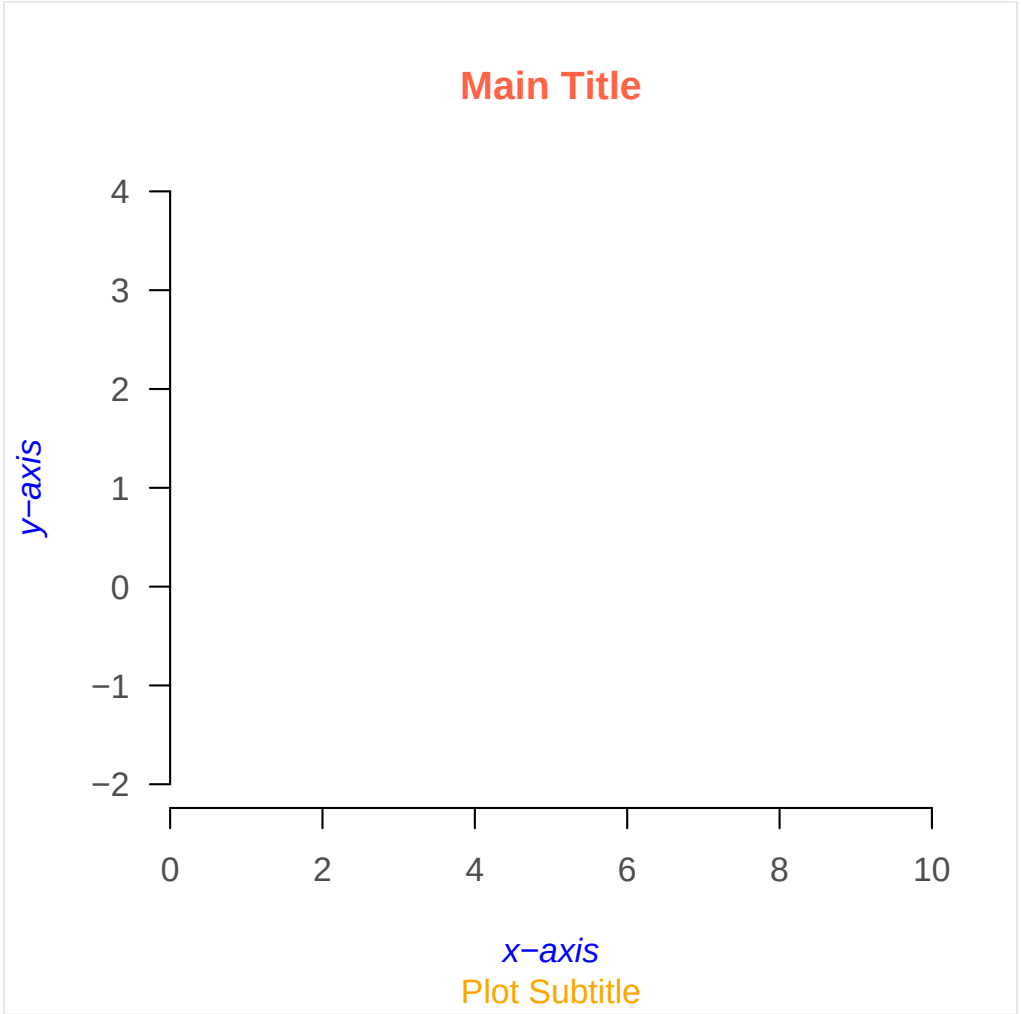
It is also possible to create a plot from scratch. Although this procedure is less documented, it is extremely flexible:

1. call `plot.new()` to start a new plot frame
2. call `plot.window()` to define coordinates
3. then call low-level functions:
4. typical options involve `axis()`
5. then `title()` (title, subtitle)
6. after that call other function: e.g. `points()`, `lines()`, etc

Plot from Scratch

```
plot.new()
plot.window(xlim = c(0, 10), ylim = c(-2, 4), xaxs = "i")
axis(1, col.axis = "grey30")
axis(2, col.axis = "grey30", las = 1)
title(main = "Main Title",
      col.main = "tomato",
      sub = "Plot Subtitle",
      col.sub = "orange",
      xlab = "x-axis", ylab = "y-axis",
      col.lab = "blue", font.lab = 3)
box("figure", col = "grey90")
```

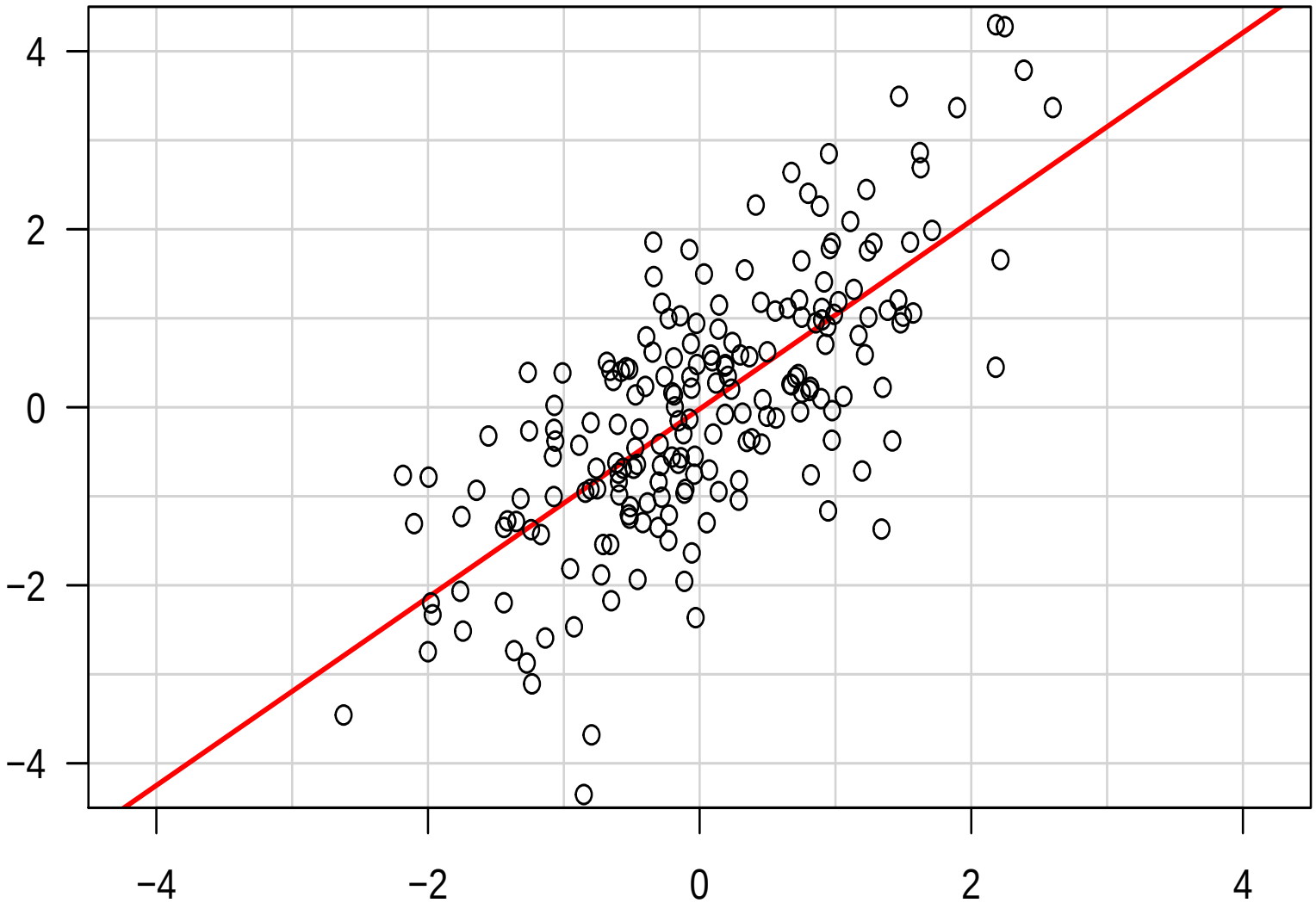
Plot from Scratch



Another plot from scratch

```
set.seed(5)
x <- rnorm(200)
y <- x + rnorm(200)

plot.new()
plot.window(xlim = c(-4.5, 4.5), xaxs = "i",
            ylim = c(-4.5, 4.5), yaxs = "i")
z <- lm(y ~ x)
abline(h = -4:4, v = -4:4, col = "lightgrey")
abline(a = coef(z)[1], b = coef(z)[2], lwd = 2, col = "red"
points(x, y)
axis(1)
axis(2, las = 1)
box()
title(main = "A Fitted Regression Line")
```

A Fitted Regression Line



Creating a Plot from Scratch

- Start a new plot with plot.new()‘
- `plot.new()` opens a new (empty) plot frame
- `plot.new()` chooses a default plotting region

Setting Up Coordinates

Then use `plot.window()` to set up the coordinate system for the plotting frame

```
# axis limits (0,1)x(0,1)
plot.window(xlim = c(0, 1), ylim = c(0, 1))
```

By default `plot.window()` produces axis limits which are expanded by 6% over those actually specified.

Setting Up Coordinates

The default limits expansion can be turned-off by specifying `xaxs = "i"` and/or `yaxs = "i"`

```
plot.window(xlim, ylim, xaxs = "i")
```

Aspect Ratio Control

Another important argument is `asp`, which allows us to specify the *aspect ratio*

```
plot.window(xlim, ylim, xaxs = "i", asp = 1)
```

`asp = 1` means that unit steps in the x and y directions produce equal distances in the x and y directions on the plot. (Important to avoid distortion of circles that look like ellipses)

Drawing Axes

The `axis()` function can be used to draw axes at any of the four sides of a plot.

- `side=1` below the graph
- `side=2` to the left of the graph
- `side=3` above the graph
- `side=4` to the right of the graph

Customizing Axes

Axes can be customized via several arguments (see `?axis`)

- location of tick-marks
- labels of axis
- colors
- sizes
- text fonts
- text orientation

Plot Annotation

The function `title()` allows us to include labels in the margins

- `main` main title above the graph
- `sub` subtitle below the graph
- `xlab` label for the x-axis
- `ylab` label for the y-axis

Customizing Annotations

The annotations can be customized with additional arguments for the fonts, colors, and size (expansion)

- `font.main, col.main, cex.main`
- `font.sub, col.sub, cex.sub`
- `font.lab, col.lab, cex.lab`

Drawing Arrows

Arrows can be drawn with the function:

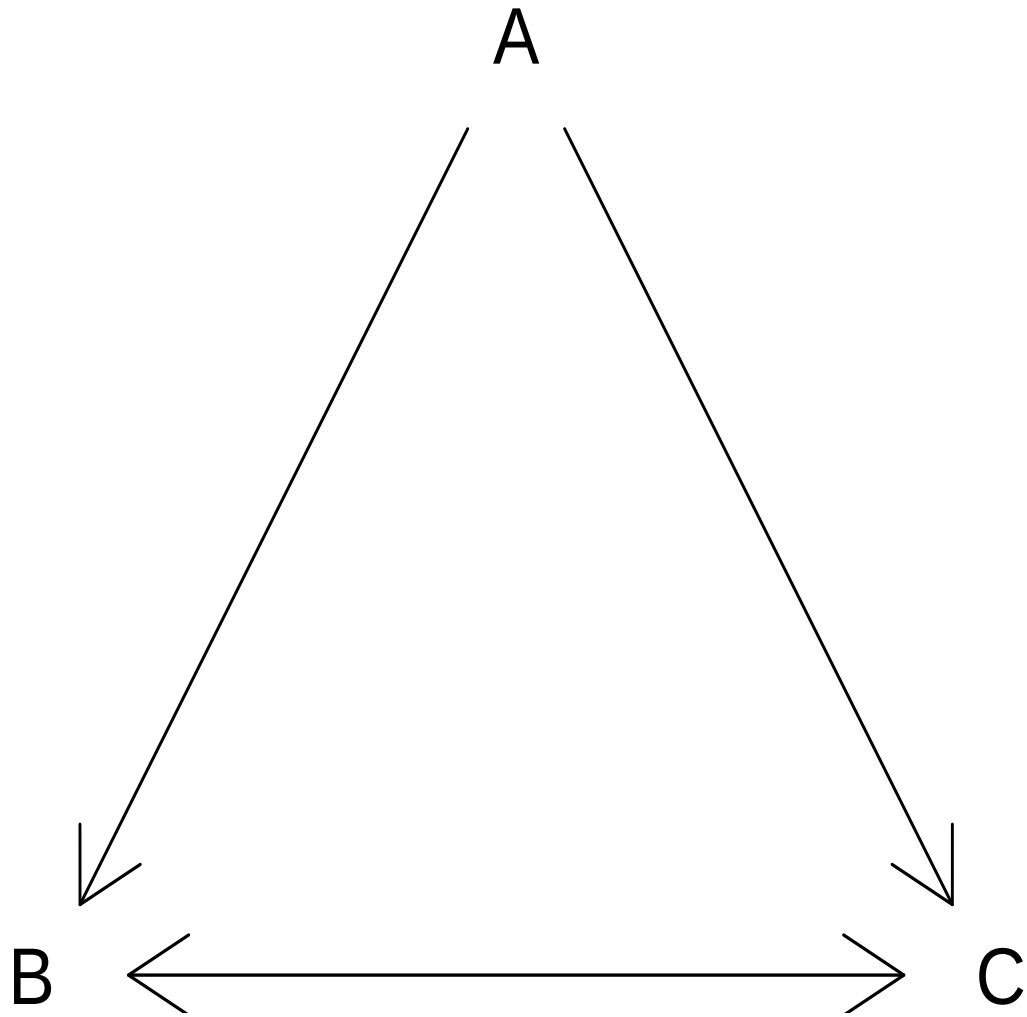
```
arrows(x0, y0, x1, y1, code = int,
       length = num, angle = num)
```

- The `x0, y0, x1, y1` arguments give the start and end coordinates.
- `code=1` head at the start, `code=2` head at the end, `code=3` head at both ends
- `length` of the arrow head and `angle` to the shaft

Drawing Arrows

```
plot.new()
plot.window(xlim = c(0, 1), ylim = c(0, 1))
arrows(.05, .075, .45, .9, code = 1)
arrows(.55, .9, .95, .075, code = 2)
arrows(.1, 0, .9, 0, code = 3)
text(.5, 1, "A", cex = 1.5)
text(0, 0, "B", cex = 1.5)
text(1, 0, "C", cex = 1.5)
```

Drawing Arrows



Drawing Rectangles

Rectangles can be drawn with the function:

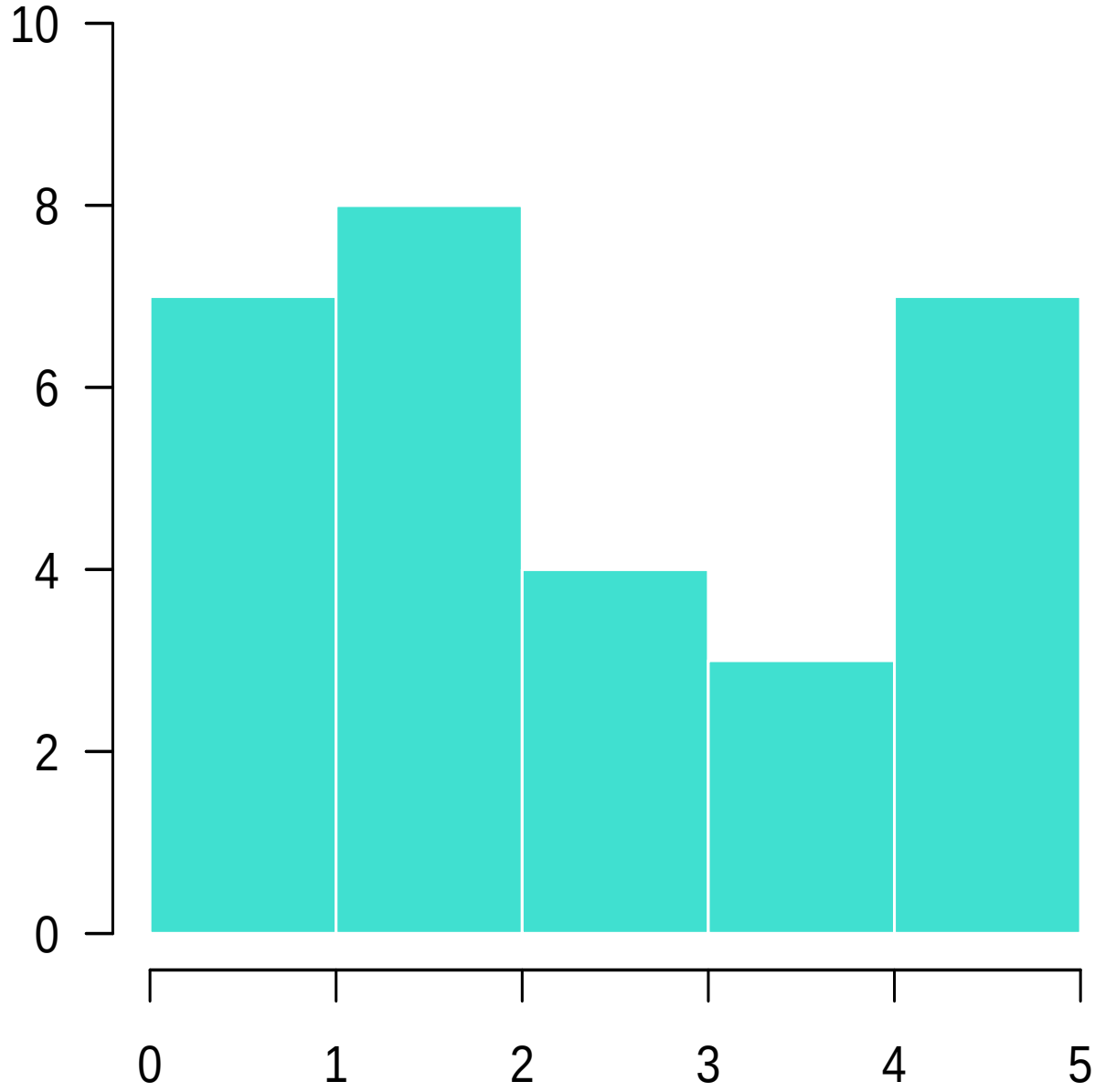
```
rect(x0, y0, x1, y1, col = str, border = str)
```

- `x0, y0, x1, y1` give the coordinates of diagonally opposite corners of the rectangles.
- `col` specifies the color of the interior.
- `border` specifies the color of the border.

Drawing Rectangles

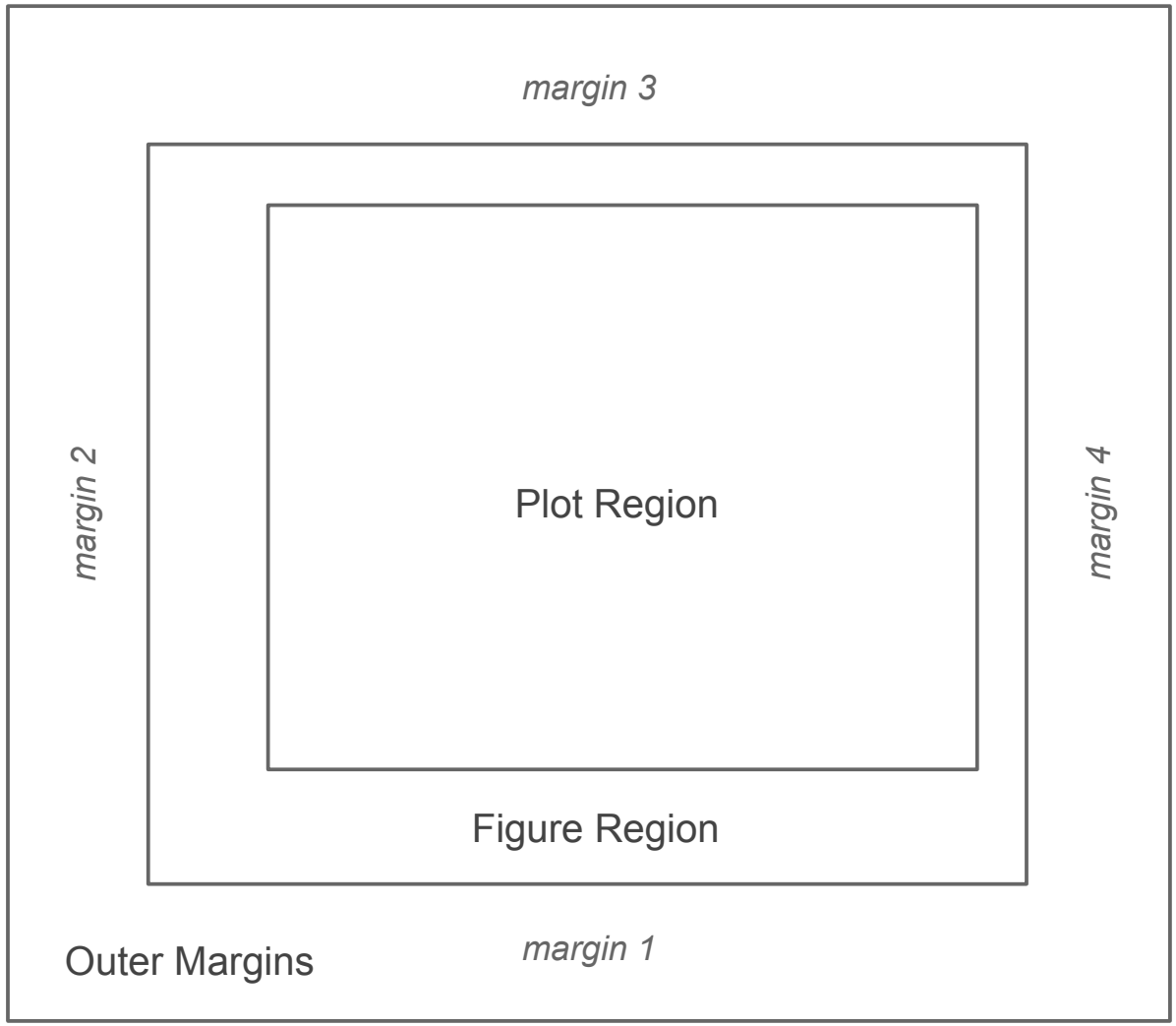
```
# barplot manually constructed
plot.new()
plot.window(xlim = c(0, 5), ylim = c(0, 10))
rect(0:4, 0, 1:5, c(7, 8, 4, 3),
     col = "turquoise",
     border = "white")
axis(1)
axis(2, las = 1)
```

Drawing Rectangles

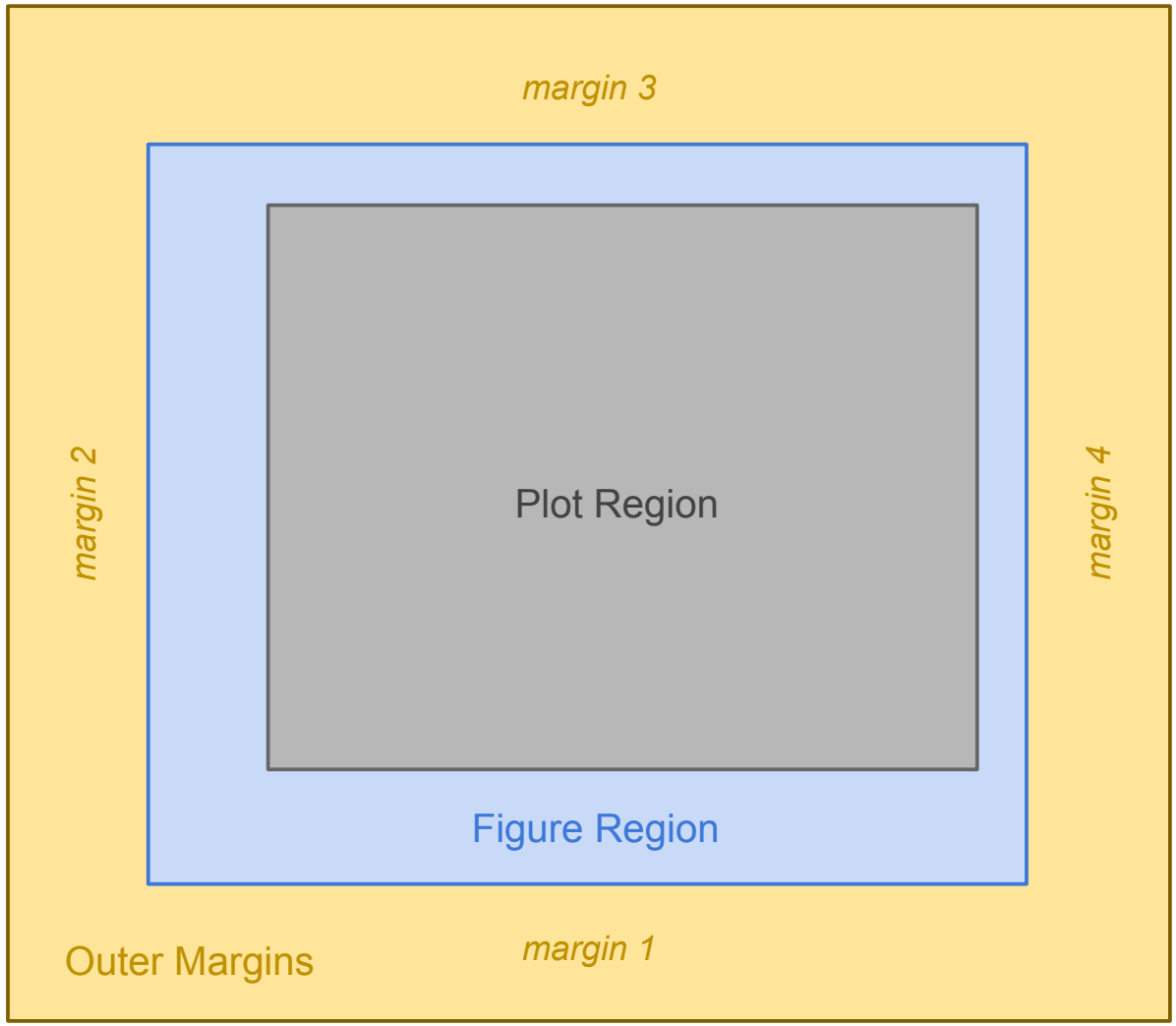


Plot Regions

Anatomy of Plot Frame and Region



Anatomy of Plot Frame and Region



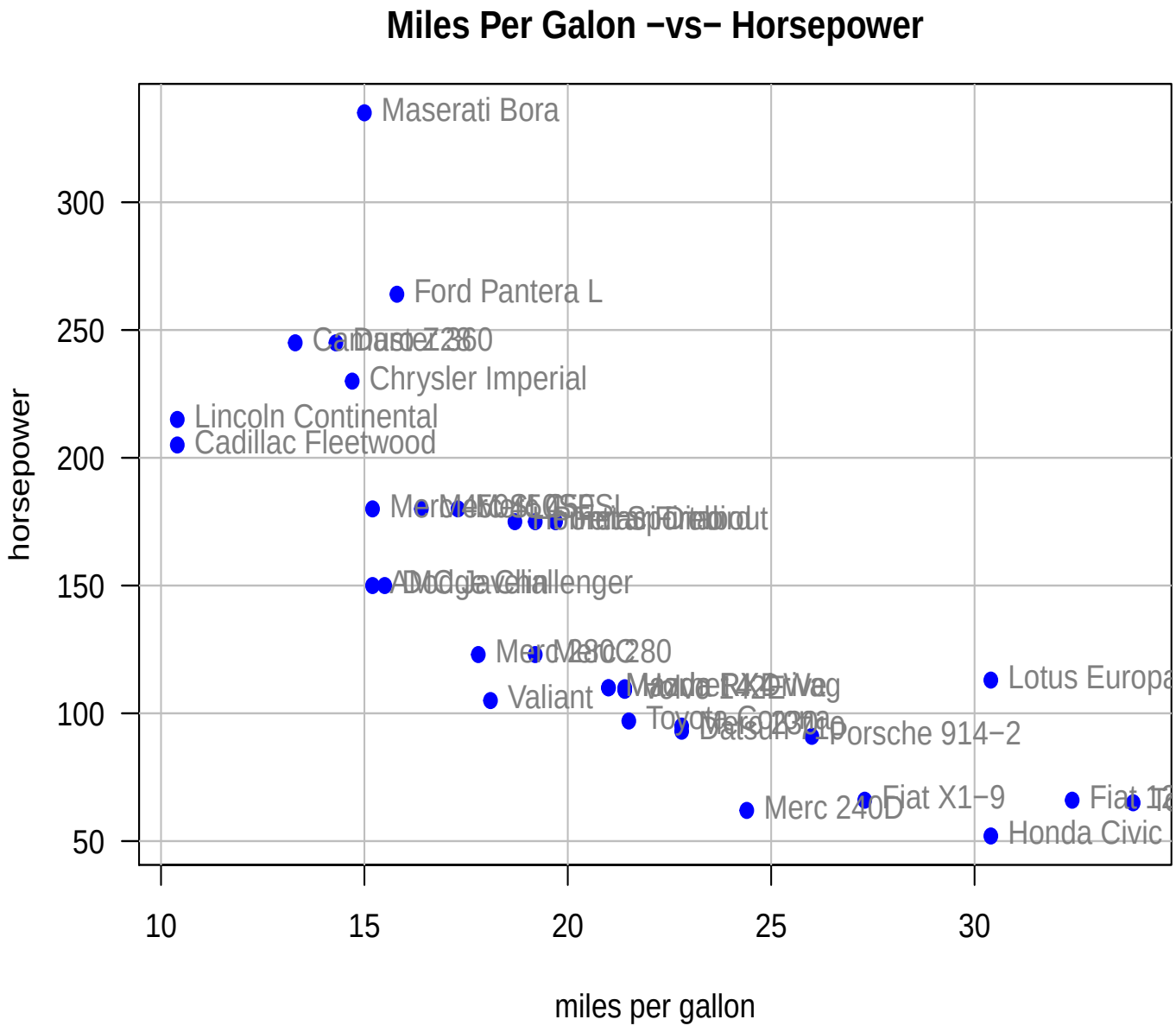
Adjusting the Margins

Margins can be adjusted with the `par()` function in various ways:

- In inches: `par(mai = c(2, 2, 1, 1))`
- In lines of text: `par(mar = c(4, 4, 2, 2))`
- Width and Height in inches: `par(pin = c(5, 4))`

One more scatter plot

```
# simple scatter-plot
op <- par(mar = c(5, 4, 3, 1))
plot(mtcars$mpg, mtcars$hp, type = "n", las = 1,
     xlab = "miles per gallon", ylab = "horsepower")
# grid lines
abline(v = seq(from=10, to=30, by=5), col = 'gray')
abline(h = seq(from=50, to=300, by=50), col = ' gray')
# points
points(mtcars$mpg, mtcars$hp, pch = 19, col = "blue")
# text (point labels)
text(mtcars$mpg, mtcars$hp, labels = rownames(mtcars),
     pos = 4, col = "gray50")
# title
title("Miles Per Galon -vs- Horsepower")
# reset graphical margins
par(op)
```


Miles Per Galon −vs− Horsepower
horsepower
miles per gallon
300
250
200
150
100
50
10
15
20
25
30
Maserati Bora
Ford Pantera L
Chrysler Imperial
Lincoln Continental
Cadillac Fleetwood
Valiant
Porsche 914−2
Fiat X1−9
Merc 240D
Honda Civic
Lotus Europa